Photo by WestBoundary Photography chris gill on Unsplash

3 Simple Reasons Why You Should First Visualize Data Before Doing Anything Else

90% of the information processed by our brain is visual, and our brain processes visuals thousands of times faster than texts. Why not take advantage of it?

Yong Cui, Ph.D.
Apr 8 · 8 min read ★

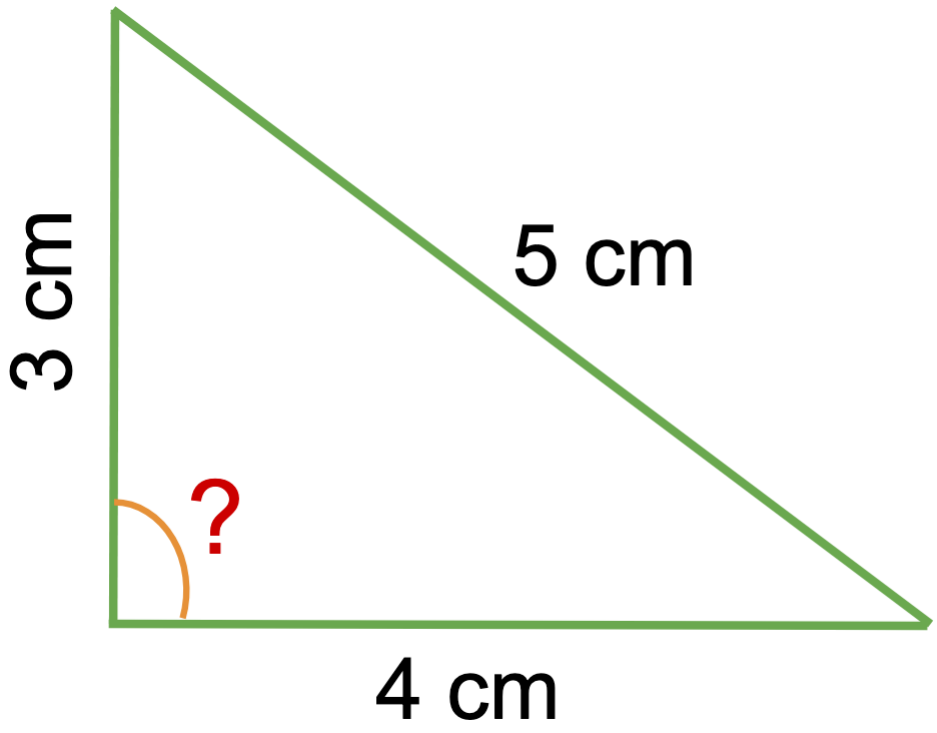
Introduction

When I was solving geometric problems in middle school, the first thing that I always did was to visualize the available data, especially

hypothetical problem.

> *The lengths for a triangle's sides are 3 cm, 4 cm, and 5 cm, respectively. How many degrees is the largest angle?*
>
> ***A.*** *85 degrees*
> ***B.*** *90 degrees*
> ***C.*** *95 degrees*

For elementary students starting their 4th or 5th grade, they presumably don't know the Pythagorean theorem yet. Their best bet to solve this problem is to draw a triangle that measures 3, 4, and 5 cm precisely (something as below hopefully) and then measure the largest angle. They should be able to find out that the largest angle is 90 degrees.



Triangle With 3, 4, and 5 cm Sides — Author

Doing data science is not dealing with just three data points. With

more complex data, visualizing the data is even more crucial to make sense of our data. Here, I've seen many great articles that introduce various handy tools (e.g., Tableau and Python modules like `matplotlib` and `seaborn`) for data visualization and various kinds of fancy graphs (e.g., heat maps, spider charts) for different scenarios, mostly for result reporting purposes.

Surprisingly, as far as I know, little has been discussed on the importance of data visualization at the early stage of data science projects. The lack of extensive discussions can be due to 1) seasoned data scientists take it as a routine step that they don't bother talking about, and 2) fresh data scientists haven't probably realized the necessity of visualizing data as a crucial pre-processing step.

In the article, I would like to share with you three key reasons why we should visualize our data before starting any subsequent data processing and analysis steps. Here are four things to note before we start.

- To provide a proof of concept, I'm going to use synthetic data that simulate pertinent data patterns that we may or may not encounter with real-world data. Please feel free to use your data if you prefer. BTW: you can find the code used in this article on GitHub.
- Visualization only makes sense when the resolution is reasonable. So with an enormous dataset, you'll have to run some iterations to review the data segment by segment to produce sharp graphs.
- This article is intended for beginner to intermediate level data scientists. Knowledge using Python and its related modules (e.g., Pandas) is preferred, but the philosophies covered here still hold using other data processing and visualization tools.

- This article isn't exhaustive. For each reason discussed below, there are other visualization approaches to attain the same objectives. Besides, there are other reasons we should visualize our data before running any analysis or building our models.

. . .

1. Understand the Missingness Pattern

Real-world data have the missingness problem for various reasons. Subjects refuse to answer specific questions due to privacy concerns like race and income. Subjects accidentally enter the incorrect data (e.g., missing a digit in the phone number). In longitudinal research, some subjects drop out of the studies due to withdrawal or death.

Thus, it's essential to understand the missingness of our data, especially those collected from the real-life studies. Indeed, we can count the missing data with descriptive statistics, but visualizing the data is a robust and straightforward means to generate an overall impression of the data's missingness.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Generate a random data set

arr = np.random.randn(50, 100)
# Inject some missing data mimicing data entry mistakes, subjects not reporting, etc.
arr[abs(arr) > 2.0] = np.nan
df_missing = pd.DataFrame(arr)
```

```
# Attrition, missing data after a certain timepoint
df_missing.iloc[6, 9:] = np.nan
df_missing.iloc[10, 70:] = np.nan
df_missing.iloc[13, 30:] = np.nan
df_missing.iloc[28, 21:] = np.nan
# Data collection errors resulting some data not collected systematically
df_missing.iloc[:, 7] = np.nan
df_missing.iloc[:, 18] = np.nan
```

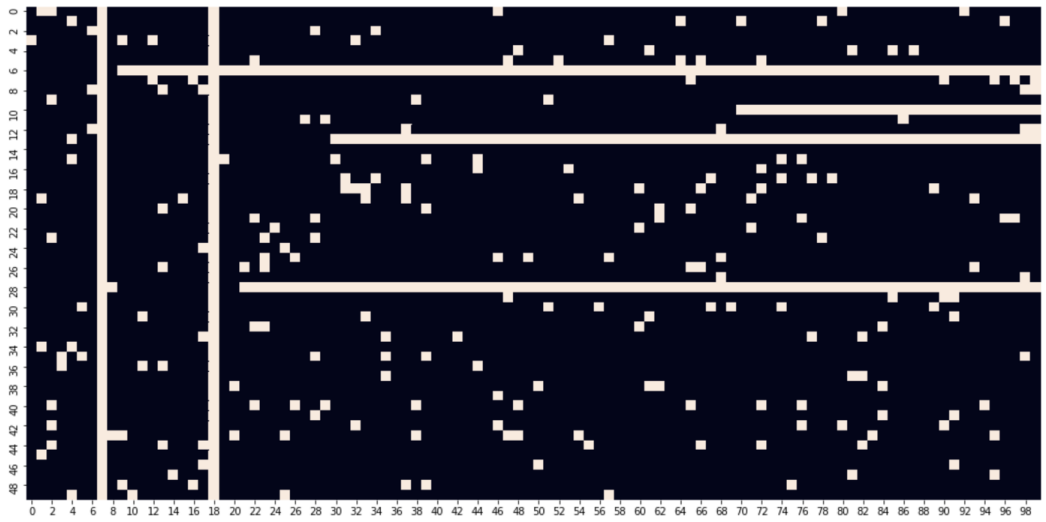
Generation of Synthetic Data

In the above code snippet, we generated a DataFrame of random

In the above code snippet, we generated a DataFrame of random numbers. We then injected some missing data. The easiest way to know the overall missingness pattern is to use a heat map, as shown below. Each cell in the light beige color represents a missing value.

```
plt.figure(figsize=(18, 9))
sns.heatmap(df_missing.isnull(), cbar=False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x12cc3bfd0>
```



Missingness Pattern Revealed by Heat Map

We can have the following observations based on this heat map. 1) Overall, the missingness of this simulated dataset is not bad. To find out the exact counts for each variable, we can use some descriptive statistics. 2). If each row represents a subject's data in a longitudinal study, it appears that some subjects drop out of the study at different time points. We can cross-check with their data charts. 3). Two columns have missing data for all subjects, which are very concerning. The staff may have misunderstood the protocol and thus not collected these data as they were supposed to. Alternatively, it's possible that these data haven't yet been entered into the database.

Visualizing the missing data with this simple heat map helped us understand the missingness pattern of this dataset, **which, more importantly, could lead us to go back to the data source to recover some of the missing data.** In the end, we can have less missingness in the data — we all know how crucial a complete dataset means to any data science projects.

. . .

2. Identify Outliers

One early step in data pre-processing is **to handle outliers, which are data values that are distinctly different from other data values in the dataset**. They can happen at least for two reasons. First, the data are entered by mistake. For example, missing a decimal can make the value 100 times larger (10.24 vs. 1024). Second, the dataset has true outliers. For instance, in a physiology study in a healthy population, most of the subjects have a resting heartbeat rate of 60 –100 bpm, but for some reason, a subject has 130 bpm, which can be a true outlier.

We'll know the negative impacts of outliers on data analysis. Here's a plausible real-life example, on which I don't know whether it has been studied yet or not though. Consider studying the impact of dropping out of college on one's success in business. We can probably incorrectly conclude that dropping out of Harvard could significantly increase your likelihood of becoming multi-billionaires than completing your four years at Harvard. Why could it happen? It's all because both Bill Gates and Mark Zuckerberg, two true outliers, dropped out of Harvard.

All joking aside, data visualization is a powerful way to identify the possible outliers in our datasets. Depending on the nature of the data structures, there are different types of charts that can visualize outliers. Here, I'm just showing you some common ones.

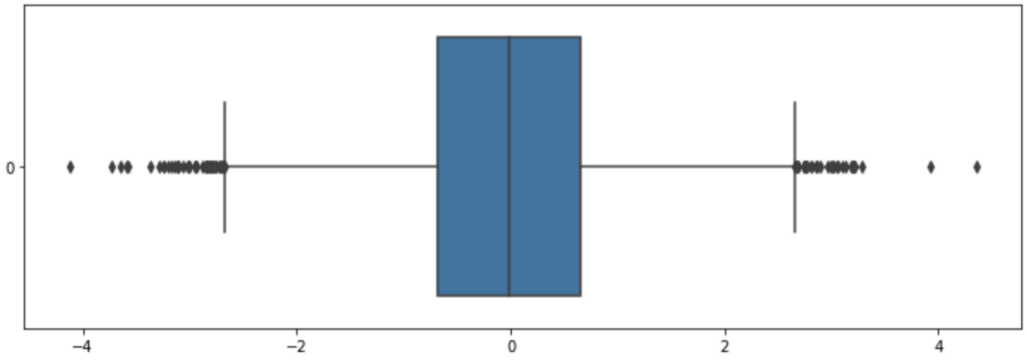
To visualize outliers using a univariate approach, the box plot is applicable in most cases. As shown in the figure below, I generated 10,000 random numbers with a normal distribution. Using the box plot, we can see that this dataset has outliers that we'll have to deal with. And handling outliers can be another topic for one article in the future.

article in the future.

```
arr_boxplot = np.random.randn(10000)
```

```
plt.figure(figsize=(12, 4))
sns.boxplot(data=arr_boxplot, orient='h')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x131133780>
```



Box Plot

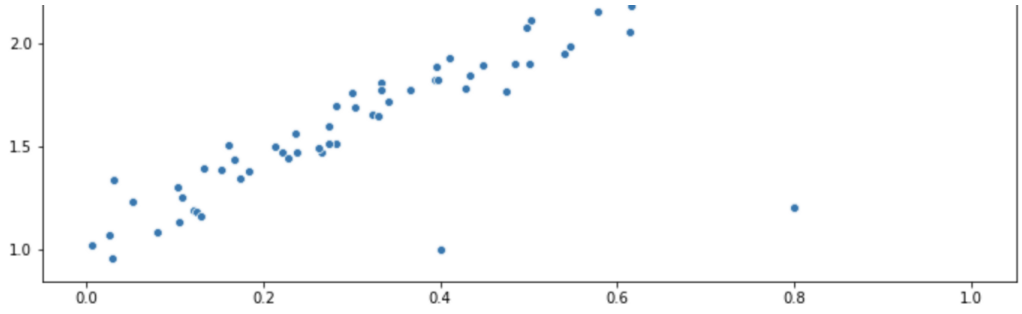
If we're to study the relationship between two continuous variables, we can use the scatter plot in most cases. As shown below, I generated two one-dimensional arrays of random numbers with a projected linear relationship between them. I artificially added some presumable outliers. As you can see, these four outliers are evidently apart from other points in the scatter plot. Please note that with a real-world dataset, if it's possible, we need to go back to the data source to verify that these data are valid values that are collected without any human mistakes.

```
# Generate random numbers that have linear relationships
x = np.random.rand(100)
y = 2 * x + 1 + .1 * np.random.randn(100)
```

```
# Aritifically set some outliers
x[5:9] = [0.2, 0.4, 0.6, 0.8]
y[5:9] = [2.5, 1.0, 3.0, 1.2]
```

```
plt.figure(figsize=(12, 6))
sns.scatterplot(x=x, y=y)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x12ce67eb8>
```



Scatter Plot

. . .

3. Generate Meaningful Hypotheses

In data science or the neuroscientific research that I'm doing, it's critical to develop some testable hypotheses with available data. Similarly, in a machine learning setting, it's always necessary to form some hypotheses and develop some meaningful models with the training dataset, which we hope hold when we run the models in the test dataset.

Similar to the identification of outliers, specific data structures of given datasets will determine how we visualize the data before doing any testing. For example, if you're trying to identify a possible linear relationship between two continuous variables, you can start with the scatter plots, as shown in the last section. Since you've just seen that, I'll show you two other types of plots to generate other potential hypotheses for particular research needs.

If you're trying to identify a trend in longitudinal research, the best chart to start with is the line chart. In the example below, I downloaded the tobacco use data from Kaggle. For simplicity, I only included the data for the nationwide average in the categories of daily and former smokers. Because the percentage data were read as strings, I converted them to floats for plotting purposes. From the line plot, we can observe a trend of decreasing prevalence of daily smoking, while the prevalence of former smokers stayed. By

identifying this trend, we could probably generate a hypothesis that the daily smoking prevalence would continue to drop after 2010.

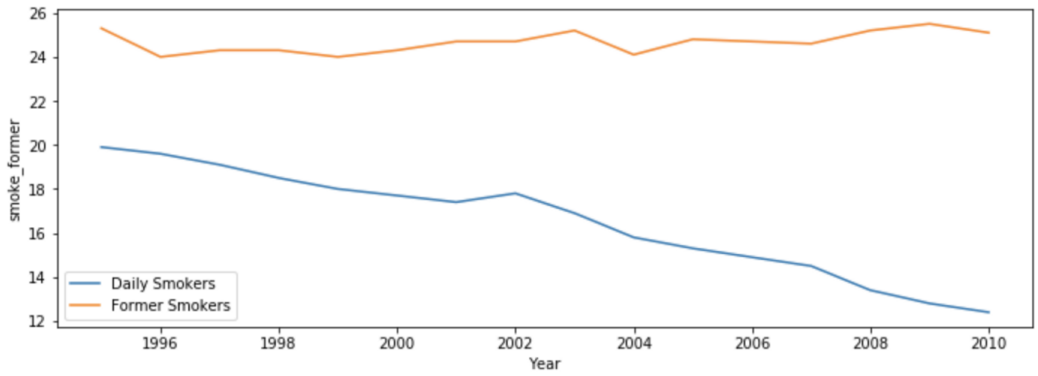
```
# Read the data
df_tobacco = pd.read_csv('tobacco.csv')
# Check the data information
# df_tobacco.info()
# df_tobacco.head()
# df_tobacco["State"].unique()
```

```
# Select the data that are the national averages
national_only = df_tobacco["State"] == "Nationwide (States and DC)"
cols = ["Year", "Smoke everyday", "Former smoker"]
df_national = df_tobacco.loc[national_only, cols].copy()
```

```
# Recode the percentage string data to float
df_national["smoke_everyday"] = df_national["Smoke everyday"].map(lambda x: float(x[:-1]))
df_national["smoke_former"] = df_national["Former smoker"].map(lambda x: float(x[:-1]))
```

```
# Plot the rate over the years
plt.figure(figsize=(12, 4))
sns.lineplot(x=df_national["Year"], y=df_national["smoke_everyday"], label="Daily Smokers")
sns.lineplot(x=df_national["Year"], y=df_national["smoke_former"], label="Former Smokers")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x12d2ecba8>
```



Line Plot

If you're trying to study the relationship between two variables (continuous and/or categorical), several options are available, such as scatter plots (the one we used in the last section for linear relationship), heat maps, and bar plots. For the sake of showing you something different, I would like to show you the swarm plots, which are useful to examine the relationship between categorical and continuous variables, with the latter's distribution being shown.

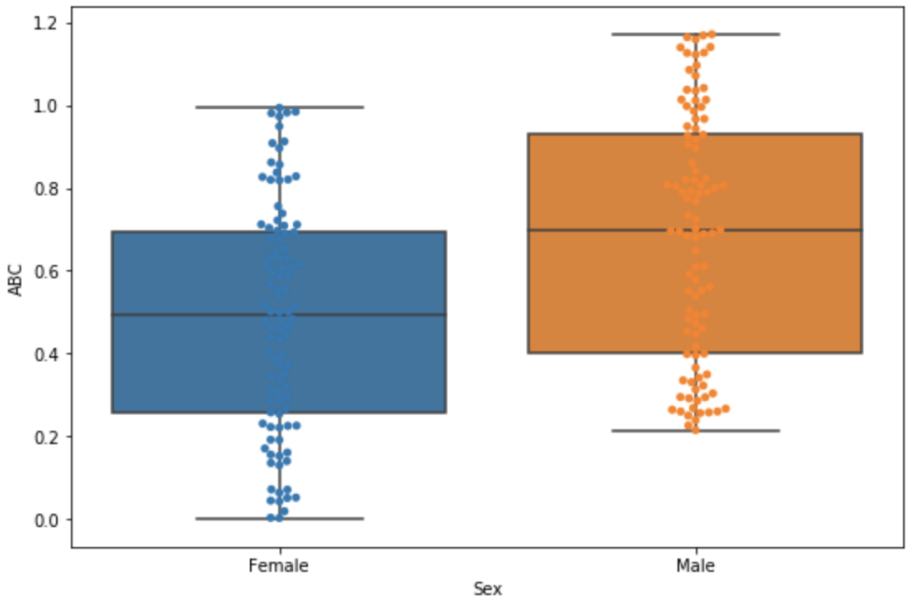
Typical examples include comparing the difference of a particular measure between two groups and comparing the difference before

and after a procedure of the same group. Pretty much, it's like something for independent and dependent t-tests.

```
# Generate some simulation data
abc_female = pd.Series(np.random.rand(100), index=['Female']*100, name='ABC')
abc_male = pd.Series(np.random.rand(100) + 0.2, index=['Male']*100, name='ABC')
abc = pd.concat([abc_female, abc_male], axis=0).reset_index()
abc.rename(columns={'index': 'Sex'}, inplace=True)
# Check data information
# abc.head()
```

```
# Generate the plot
plt.figure(figsize=(9, 6))
sns.boxplot(x="Sex", y="ABC", data=abc, whis=np.inf)
sns.swarmplot(x="Sex", y="ABC", data=abc)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x12e5d8a58>
```



Swarm Plot & Box Plot

Suppose that we want to study whether sex (a biological factor used in biomedical research instead of using gender, which has some social implications) impacts the particular severity measure ABC of COVID-19. The above diagram shows the data simulation and generation of the swarm plot mixed with the box plot using the `seaborn` module. With this figure, we can hypothesize that men may have higher levels of the ABC factor than women, a hypothesis that we can test with a larger population later.

• • •

Conclusions

Data pre-processing is an art, which separates topmost data scientists from mediocre ones. Some key components in the data pre-processing steps are the identification and handling of the missing values and outliers. We have to eyeball the data to make sure the data themselves have good quality. If conditions are allowed, it's always a good idea to go back to the data source to verify data integrity and eliminate human mistakes at the source.

In addition to checking the text description of our data, it's incredibly instrumental in visualizing the data in the early stage of our data processing steps rather than saving the visualization tools only for later showcasing the ending products. Such visualization provides a fundamental way to inform the data quality and generate possible